Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Maj Smerkol

Digitalni video efekti za After Effects

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor:
izr. prof. dr. Narvika Bovcon

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi preglejte uporabo vizualnih učinkov v medijih gibljivih slik. Načrtujte in razvijte vtičnike za Adobe After Effects, s pomočjo katerih bo mogoče na določen način spremeniti video podobo na ravni posameznih slik.

Zahvaljujem se prijateljem in družini za podporo pri delu in Heleni Pivec za pomoč. Zahvaljujem se tudi profesorju Patriciju Buliću za mentorstvo v prejšnjem letu in seveda mentorici Narviki Bovcon.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
VFX	visual effects	vizualni učinki
SFX	special effects	posebni učinki
AEGP	After Effects general purpose plug-in	splošno namenski vtičnik za After Effects
AE	Adobe After Effects	Adobe After Effects
SDK	software development kit	paket za razvoj programske opreme
CGI	computer generated imagery	računalniško generirana grafika

Povzetek

Naslov: Digitalni video efekti za After Effects

Avtor: Maj Smerkol

Razvili smo tri različne vtičnike za Adobe After Effects. Vsak implementira originalen vizualni učinek. Pred tem smo opravili pregled vizualnih učinkov. Ob tem se nismo omejili na digitalne, ampak tudi vizualne učinke za film in video. Moderni digitalni video zaradi visoke kvalitete slike in velike fleksibilnosti omogoča oba pristopa. Digitalni vizualni učinki se danes uporabljajo v filmskem svetu in svetu avtorskega videa.

Razvili smo tri vtičnike za vizualne učinke. Z vtičnikom PixelRain lahko izdelamo nadzorovane vizualne artefakte. Z vtičnikom ElStAberr lahko sliki dodamo globino ali preoblikujemo vizualne elemente. Vtičnik MirrorMosaic je namenjen izdelavi ponavljajočih se video vzorcev.

Digitalne vizualne učinke smo razvili kot vtičnike za programski paket After Effects. Ta ponuja paket za razvoj programske opreme, ki uporabniku ponudi zmogljiv programski vmesnik. Med delom se je izkazalo, da je za zmogljivost in pravilnost delovanja najpomembnejše pravilno delo s pomnilnikom. Razviti vtičniki so uporabno orodje za oblikovanje vizualnih učinkov.

Ključne besede: video, After Effects, vizualni učinki, posebni učinki.

Abstract

Title: Digital video effects for After Effects

Author: Maj Smerkol

We developed three different plug-ins for Adobe After Effects. Each of them implements an original visual effect. Before that we made an overview of visual effects. We didn't limit ourselves to digital ones, but also visual effects for film and video. Digital visual effects are used in the world of cinema as well as the world of video.

We developed three VFX plug-ins. Plug-in PixelRain can be used to create controlled visual glitches. Using plug-in ElStAberr we can add depth to image or transform visual elements. MirrorMosaic's intended use is to create looping video patterns.

The visual effects were created as plug-ins for the After Effects software. It offers a package for software development, which offers the user a capable program interface. While working on the effects it turned out that the most important aspect in this kind of work is good working memory efficiency. The final plug-ins are a useful tool for creating visual effects.

Keywords: video, After Effects, visual effects, special effects.

Poglavje 1

Uvod

Cilj diplomske naloge je raziskati metode razvoja vtičnikov za programsko orodje Adobe After Effects (v nadaljevanju AE). AE je programski paket, namenjen montaži videa in izdelavi vizualnih učinkov. Ena izmed njegovih dobrih lastnosti je razširljivost s pomočjo vtičnikov, ki omogočajo dodajanje poljubnih funckionalnosti programskemu paketu. Razširjanje je omogočeno prek programskega vmesnika in prosto dostopne razvojne dokumentacije za uporabo le tega.

Vtičnike za AE lahko delimo v štiri kategorije: splošno namenske vtičnike (AEGP), vizualne učinke (effect), vhodno izhodne vtičnike (IO) in vtičnike Artisan, ki prevzamejo upodabljanje 3D slojev oziroma nadomestijo privzeti algoritem za upodabljanje v AE. Gostitelj še vedno sam skrbi za upodabljanje 2D slojev. Ker sem se odločil za razvoj vizualnih učinkov, je izbira med tipi vtičnikov jasna. Vtičniki za vizualne učinke implementirajo VFX kot operacijo nad posameznimi sličicami v videu. Programski vmesnik AE ponuja nemalo operacij, ki pospešijo izvajanje operacij nad piksli in skrajšajo čas razvoja, saj ponujajo visokonivojski dostop do vgrajenih funkcij AE.

Splošno namenski vtičniki zahtevajo več resursov od sistema, na katerem tečejo, omogočajo pa programerju več možnosti integracije z AE, spreminjanje uporabniškega vmesnika in odzivanje na interno stanje celotnega programskega paketa. Omogočajo mnogo več, kot potrebujemo za imple-

mentacijo tipičnih vizualnih učinkov. Vhodno izhodni vtičniki so namenjeni dodajanju podpore za branje in pisanje novih datotečnih formatov, implementaciji kodekov in podobno.

Pri razvoju VFX je seveda prvi korak ugotavljanje namena vizualnega učinka oziroma definiranje umetniške vizije. Pred tem sem pregledal obstoječe vtičnike. Po namenu sem jih ločil v tri kategorije. Prva kategorija so vtičniki za popravljanje izvornih posnetkov, kot so odstranjevanje šuma, popravljanje raznih vizualnih artefaktov in podobno. Namen teh vtičnikov je reševanje slabih posnetkov, kadar ponovni zajem videa oziroma ponovno snemanje ni mogoče, zaradi cene, pomanjkanja časa ali drugih razlogov. Le redko lahko s pomočjo teh vtičnikov dosežemo rezultate enake kvalitete, kot če bi bili posnetki dobri.

V drugo kategorijo štejem efekte, ki so namenjeni dodajanju vrednosti posnetku v post produkciji. Večino vizualnih učinkov pravzaprav lahko štejemo v to kategorijo. Sem spadajo tudi vsi tisti efekti, ki vizualno pomagajo pripovedi, pomagajo ustvariti fiktivni svet, like in podobno.

V tretjo kategorijo uvrščam destruktivne efekte - to so tisti, ki ne dodajajo informacij sliki, ampak nasprotno zmanjšajo kvaliteto slike za dosego nekega vizualnega sloga ali pa pomagajo ustvarjati vzdušje in gledalcu, če so uspešno uporabljeni, lahko vzbudijo določena čustva. Primere takšnih efektov lahko najdemo marsikje, v popularni kulturi (Mr Robot, 2015), računalniških igrah (SOMA, 2015) in filmih (The Hunger, 1983), na različne načine proizvedene vizualne artefakte pa celo v galerijah in knjižnih zbirkah umetniških del (ang. *ASCII Graphic Glitch Art*, Rozita Fogelman).

Poglavje 2

Vizualni učinki v filmu in videu

2.1 Kaj so vizualni učinki in zakaj jih uporabljamo

Vizualni učinki so različne metode, kako sliki (filmu, videu, fotografiji ali drugačnemu vizualnemu gradivu) dodamo elemente oziroma spremenimo ali izboljšamo sliko[10]. Gre za tehnike, ki jih ne moremo posneti s kamero med snemanjem žive akcije. Uporabljamo jih iz različnih razlogov, predvsem pa kadar bi bilo želeno nemogoče posneti, kot zahteva scenarij (primer potovanje na luno v filmu *Apollo 13*, 1995). Če bi bilo snemanje prizora prenevarno za igralce ali druge udeležence na snemanju, se prav tako lahko uporabi vizualne učinke za dosego končnega rezultata brez izpostavljanja ljudi pravi nevarnosti. Tako so uporabljeni vizualni efekti naprimer v akcijskih filmih, na primer v filmu *Fire Brigade*, 1926, v katerem je prikazana dojenčica v goreči hiši. Naslednji razlog za uporabo vizualnih efektov pa je zniževanje cene produkcije. Tako naprimer v filmu Gospodar prstanov: Stolpa (ang. *The Lord of the Rings: The Two Towers*, 2002) horde orkov napadajo trdnjavo. Najeti toliko statistov bi bilo drago in nepraktično, saj je vsak moral tudi čez proces maske vsak snemalni dan. Lažje in prav tako dobro je posneti manjšo skupino maskiranih statistov in jih nato v postprodukciji vstaviti v sliko večkrat.

Med vizualne učinke štejemo na vse načine pridobljene grafične elemente, ki jih ne moremo posneti hkrati z akcijo. Ločimo jih od posbnih učinkov oziroma praktičnih učinkov. Posebni učinki pa so tisti, ki jih posnamemo med snemanjem akcije. Primeri takšnih učinkov so streli z orožjem, eksplozije, umetni dež in podobno. Danes se jih tipično kombinira z vizualnimi učinki za stopnjevani realizem. Primeri kombiniranih efektov so naprimer eksplozije v Transformerjih (ang. *Transformers*, 2007).

2.2 Primerjava uporabe vizualnih učinkov v filmu in videu

Ko govorimo o vizualnih učinkih, se je najenostavneje opreti na primere iz sveta filma. Filmi večinoma ciljajo na neko mero realizma. Seveda se razlikujejo po stopnji realizma - od izjemno realističnih filmov Michaela Hanekeja, naprimer Sedmi kontinent (nem. *Der Siebente Kontinent*, 1989), hitro pridemo do fantazijskih filmov in celo popolnoma neprepričljivih filmov, kot je naprimer japonski eksperimentalni film Tetsuo, železni človek (jap. *Tetsuo*, 1989). V filmu se skoraj vedno trudimo skriti vizualne učinke, jih narediti dovolj prepričljive, da jih gledalec ne opazi. Skozi leta in razvoj tehnologije so seveda vizualni učinki postajali bolj in bolj prepričljivi. Danes jih nepoučni gledalec le redko opazi, pa še takrat v nizkoproračunskih produkcijah, ki ponavadi uporabljajo cenejšo in starejšo tehnologijo, poleg tega pa ponavadi tudi nimajo dostopa do izkušenih umetnikov vizualnih učinkov.

Danes, ko je film vedno pogosteje tudi digitalno posnet in tako z vidika medija pravzaprav video, je ta razlika manjša. Z enakimi orodji delajo veliki filmski studiji in alternativni umetniki, ki ustvarjajo umetniški avtorski video. Adobejev programski paket Premiere Pro so uporabili tudi v studiju Marvel za montažo filma Deadpool (ang. *Deadpool*, 2016).

Avtorski video pa je po naravi popolnoma drugačen. V osnovi se ne opira na vizualni realizem, ampak skozi uporabo vizualnega in zvočnega materiala poskuša na človeka vplivati na čustveni ravni[1]. Pascal Bonitzer primerja

video s poezijo, specifično s haikujem. Video se odmakne od realizma prav zaradi enostavnosti digitalne manipulacije. Za razliko od filmskega traku, ki je enkratno zapisan in je za večino obdelave potrebno presnemavanje na drug trak, skeniranje v digitalno obliko in nato optično tiskanje nazaj na trak, je video v osnovi spremenljiv.

V videu so se uporabljali drugačni vizualni učinki kot v filmu. Začetek vizualnih učinkov za video predstavlja uporaba slike v sliki med napovedovanjem novic, maskiranje s pomočjo modrega zaslona pri napovedovanju vremena in podobni vizualni učinki za televizijske oddaje.

2.3 Vpliv videa na alternativno kulturo

Avtorski video je bil zelo popularen med alternativnimi umetniki v 70. in 80. letih 20. stoletja. Takrat so postale video kamere in sistemi za montažo postopoma dostopni tudi za javnost, ne le za velike televizijske in produkcijske hiše [4]. Na uporabo videa v umetnosti, sicer predvsem performativni, je močno vplival pojav Sonyjevega Portapacka leta 1967. To je bil majhen prenosen sistem za urejanje videa. Sestavljen je bil iz majhne črno-bele kamere in sistema za zapis, predvajanje in presnemavanje magnetnih trakov. Zaradi možnosti takojšnjega pregleda videa in relativno nizke cene je postal zelo popularen med protivojnimi protestniki in umetniki v ZDA, ki so bile takrat v vojni z Vietnamom [8]. Tako se je video utemeljil kot medij alternativne kulture. Seveda se je hkrati uporabljal tudi za televizijske produkcije, kar je prineslo razvoj v drugi smeri. Za namene televizijskih prenosov so bile razvite tehnike za vstavljanje slike v video, različni prehodi in podobno. Tudi takšne tehnike so bile pogosto uporabljene v umetniških produkcijah. V Sloveniji so konec 1970. let in kasneje alternativni umetniki pogosto sodelovali z RTV Ljubljana [2]. Od njih so si tudi izposojali opremo in razvili nove tehnike uporabe le te. Tehnika imenovana feedback loop (vstaljanje zakasnjenega videa samega vase) je bila razvita v alternativni sceni, kasneje pa se je uporabljala tudi v komercialnih programih.

2.4 Pregled zgodovine vizualnih učinkov

Filmska montaža je postopek, pri katerem iz posnetkov sestavimo zgodbo. Je ključna dejavnost v postprodukciji. Sergej Mihajlovič Eisenstein, ruski filmar (Oklepnica Potemkin, 1925), celo postavi montažo na prvo mesto v filmskem procesu. Pravi, da je montaža najpomembnejša faza filmske produkcije.

Zgodovinsko se je tehnologija filmske in video montaže začela razvijati že konec 19. stoletja. Leta 1895 je v kratkem filmu Usmrtitev Marije I. Škotske (ang. *The Execution of Mary, queen of Scots*, 1895) režiser Alfred Clark prikazal, kako kraljici odsekajo glavo. Navidezno usmrtitev kraljice je dosegel tako, da je tik pred rabljevim zamahom s sekiro kamero ustavil in v tem času zamenjal igralko z lutko, nato pa nadaljeval snemanje. To je prvi dokumentirani vizualni učinek. Viri navajajo, da je marsikateri gledalec verjel, da so igralko med snemanjem zares usmrtili. Ta film je pokazal, da montaža in vizualni učinki lahko pripomorejo k pripovedni moči filma.

Razvoj vizualnih učinkov gre z roko v roki z novimi montažerskimi prijemi, novo filmsko tehnologijo in razcvetom filma v 20. stoletju. Že nekaj let kasneje je za film Veliki rop vlaka (ang. *The Great Train Robbery*, 1903) studio Thomasa Edisona razvil tehnologijo maskiranja filma (ang. *black matte*) in ponovne ekspozicije maskiranega predela, da je v studiju lahko prikazal pogled skozi okno, v katerega je vstavil razgled. V 1930. letih so začeli z uporabo projekcije ozadij ali drugih elementov na platno, pred katerega so nato postavili scenografijo in igralce. Istočasno se je razvila tudi uporaba pomanjšanih setov in figur oziroma miniatur, ki so jih lahko animirali na način *stop motion* in kasneje z uporabo projekcije ali maskiranja kombinirali s posnetki igrane akcije. V tem času se je uporabljala tudi tehnika slikanja na steklo, ki se je postavilo pred lečo kamere za dodajanje vizualnih elementov. Zanimivo je, da so v 30. letih, v času črnobelih filmov, že začeli uporabljati tehniko odstranjevanja ozadja s pomočjo modrega zaslona. Moder zaslon namreč lahko z uporabo modrega filtra na črnobelem posnetku popolnoma izgine in pusti za seboj neizpostavljen film, kamor lahko vstavimo druge slike.

V 50. letih prejšnjega stoletja so v Disneyevih studijih odkrili tehniko,

kako hkrati posneti sliko in potujočo masko (ang. *traveling matte* - način maskiranja, pri katerem se masko posname na trak in se le ta zato popolnoma ujema s sliko) s pomočjo modrega zaslona. S to tehniko so posneli film Marry Poppins (1964), v katerem naslovna junakinja leti s pomočjo dežnika, oziroma s pomočjo modrega zaslona in potujoče maske[10]. Tehniko so v 80. letih izpopolnili s pomočjo na UV svetlobo občutljivega filmskega traku in barvanjem elementov maskiranja z UV odbojno barvo za film Vojna zvezd (ang. *Star Wars*, 1977).

V naslednjih desetletjih je šel razvoj filmskih posebnih učinkov v smeri izpopolnjevanja že poznanih tehnik. Tako so v 60. letih razvili novo tehniko projeciranja na zaslon v studiju od spredaj, kar je poskrbelo za bolj jasno sliko ozadij, večji kontrast in bolj pravilne barve. V 70. letih se je začela uporabljati tehnika maskiranja z modrim zaslonom tudi za barvne filme, v 80. letih pa so uspeli doseči maskiranje brez uhajanja modre svetlobe na objekte [1].

Prvi računalniško ustvarjeni vizualni efekti so se pojavili že v 50. letih 20. stoletja. John Whitney je uporabil odslužen analogni računalnik za vodenje protiletalskih izstrelkov kot pripomoček za animacijo. Na podlagi njegovega dela so kasneje razvili tudi tehniko, s katero so v filmu Odiseja 2001 posneli sekvenco potovanja skozi črvino.

Prvi pravi digitalni posebni učinki pa so bili uporabljeni v filmu Westworld (ang. *Westworld*, 1973). V tem filmu so prikazali človeku podobne robote in v nekaterih prizorih dele igralcev nadomestili s 3D računalniško generirano grafiko. Podjetje Triple-I je kasneje tehniko razvijalo dalje. Nekaj let kasneje so podobne vizualne učinke uporabili v filmu Futureworld (ang. *Futureworld*). Leta 1982 pa so za film Tron (ang. *Tron*, 1982) s pomočjo računalniške 3D CGI upodobili celoten fikcijski svet in objekte v njem. V filmu Tron efekti niso bili prepričljivi, kar pa ustvarjalcev ni oviralo, saj film prikazuje digitalen

[1] ang. *blue-spill* je odsev modre svetlobe s platna na druge objekte. Izoognemo se mu lahko s pomočjo natančne osvetlitve platna, neodvisne od drugih virov svetlobe ali s pomočjo filtrov v post produkciji.

svet oziroma navidezno resničnost. Leta 1985 je studio Pixar ustvaril prvi 3D CGI animirani lik. To je bil stekleni vitez v filmu Mladi Sherlock Holmes (ang. *Young Sherlock Holmes*). V 90. letih so digitalni vizualni učinki hitro napredovali zaradi razvoja računalniške tehnologije, ki je postala dostopnejša in končno dovolj zmogljiva za resno delo. Leta 1992 so v filmu prvič prikazali prepričljive 3D CGI živali, pingvine in netopirje v filmu Batman se vrača (ang. *Batman Returns*, 1992). Leto kasneje pa je film Jurski Park (ang. *Jurassic Park*, 1993) zelo uspešno kombiniral 3D CGI, lutke, živo igro in posnetke miniatur.

Danes velika večina filmov in videa uporablja digitalne video učinke. Ti efekti so pogosto popolnoma neopazni, kot naprimer dodani sneg v filmu Kradljivka knjig (ang. *The Book Thief*, 2013). V filmski industriji se vizualni efekti večinoma uporabljajo čimbolj neopazno in ne opozarjajo gledalca nase [10]. Izjema so namerni umetniški učinki, ki na primer v filmu Doktor Strange (ang. *Doctor Strange*, 2016) prikazujejo nek drug svet. Tam jih ne moremo zgrešiti, pa jih kljub temu gledalci lahko ignorirajo, saj so uspešno uporabljeni kot pripovedno orodje.

Poglavje 3

Uporaba After Effects za vizualne učinke

3.1 Hiter pregled After Effects

After Effects je orodje za video montažo, izdelavo vizualnih učinkov in sestavljanje videa.

3.2 Sestavljanje digitalnega videa

Primarni namen After Effects je sestavljanje (ang. *compositing*) videa. Sestavljanje je postopek, pri katerem iz več videov, slik in drugih vizualnih elementov sestavimo končno sliko tako, da jih razporedimo enega nad drugega v 2.5D prostoru[1]in nato med seboj kombiniramo[12]. Pri kombiniranju plasti lahko izberemo med več različnimi načini mešanja (ang. *blending mode*) oziroma operacijami nad piksli.

Današnja definicja sestavljanja videa je bila postavljena leta 1984 v članku Compositing Digital Images[12]. Članek je osnoval nov algoritem za sestavljanje slike s pomočjo uvedbe kanala alpha (ang. *alpha chanell*), ki ponazarja pokrivnost piksla. Kanal alpha je definiran kot relativna pokrivnost elementa, pri kateri 0 pomeni nično pokrivnost, 1 pa polno pokrivnost. Vmesne

operacija	**prevod**	**pomen**
clear	prazno	Izhodna slika je popolnoma nepokrivna oziroma prazna.
A	A	Izhodna slika je enaka prvemu operandu.
A add B	A plus B	Izhodna slika je enaka seštevku obeh operandov.
A over B	A nad B	Izhodna slika je sestavljena iz slike A, postavljene nad B. Kjer ima prvi operand vrednost alpha 1, je izhod enak prvemu operandu. Sicer je izhod linearna kombinacija ober pikslov s kanalom alpha.

Tabela 3.1: Nekaj operacij kompozicijske algebre.

vrednosti se izračunajo z linearno interpolacijo. Slike, ki vsebujejo podatek o pokrivnosti posameznih pikslov, tako brez težav lahko definiramo v prostoru RGBA.

V istem članku je bila predstavljena tudi kompozicijska algebra, kakršno uporabljamo še danes. Definira skupino operacij nad dvema slikovnima elementoma, pri čimer posebno pozornost namenja ohranjanju pravilne vrednosti kanala alpha v izhodni sliki. S tem je dosežena možnost kombiniranja mnogih slojev videa s kombiniranjem binarnih operacij med sosednjimi sloji. Operacije so naštete in razložene v tabeli.

3.3 Sestavljanje videa v After Effects

V After Effects je osnovna enota dela kompozicija (ang. *composition*)[11]. Kompozicija vsebuje slikovne plasti in učinke. Platno predstavlja delovno površino, ki tipično predstavlja ciljni format videa, ki določi njeno resolucijo,

[1] 2.5D je izraz, ki pomeni 2 dimenzionalen prostor, v katerem pa lahko posamezne 2D objekte razporedimo enega nad drugega v smeri tretje osi Z.

razmerje med višino in širino in s tem tudi relativno širino piksla. Na platnu lahko elemente postavimo v prostoru ekrana, časovnico pa uporabimo za premikanje elemntov po času. Plasti lahko tudi razporejamo po osi Z enega nad drugega.

V AE se video sestavlja na podlagi kompozicijske algebre, ki poudarja uporabnost kanala alpha. Poleg tega pa lahko dele videa tudi ločeno maskiramo oziroma izberemo predele slike, ki bodo vidni. Pri tem imamo na izbiro več orodij, kot so:

- pravokotne maske, ponavadi uporabljane za hitro odstranjevanje elementov iz sicer drugače maskiranega videa, na primer odstranjevanje luči iz videa, posnetega pred zelenim zaslonom,
- elipsoidne maske, ki se ponavadi uporabljajo v procesu popravljanja barv[2]za enostavno ločevanje obrazov od ozadja,
- rotoskopirane maske - animirane maske, ki jih uporabnik naredi s preprostim risanjem maske, AE pa mu pomaga pri ekstrapolaciji animacije na naslednje sličice,
- rotoskopirane maske s pomočjo orodja Mocca for After Effects, komercialnega produkta za rotoskopiranje mask, ki pa je v minimalni verziji že vgrajen tudi v AE,
- efektov, ki ustvarijo masko na podlagi različnih parametrov in slike, in drugih.

Tipično se orodja za hitro maskiranje uporabljajo v kombinaciji z drugimi metodami sestavljanja, na primer odstranjevanjem zelenih zaslonov in podobno. Pri vseh orodjih lahko pozicijo, velikost in obliko animiramo. To storimo tako, da na določenih sličicah masko oblikujemo, med temi ključnimi sličicami pa bo AE za nas animacijo interpoliral. Ker je interpolacija lahko nenatančna, imamo na voljo tudi urejevalnik krivulj, s pomočjo katerega lahko natančneje določamo način interpolacije.

Maskam lahko tudi nastavljamo ostrost robov. Ta parameter se v AE imenuje *feather* (slo. *pero*). Nastavimo lahko, koliko pikslov daleč od začrtane maske se bo kanal alpha interpoliral med polnim maskiranjem in odsotnosjo maske. Pri tem parametru je interpolacija linearna, saj se uporablja le kot način, s katerim masko skrijemo pred gledalcem. Ostri robovi pravilnih oblik pritegnejo pozornost, zato jih skoraj vedno vsaj malo zameglimo.

3.4 Uporaba digitalnih vizualnih učinkov v AE

Programski paket Adobe After Effects vsebuje veliko izbiro digitalnih vizualnih učinkov. Vgrajene, lastne ali druge vtičnike za učinke lahko uporabimo na več načinov. Najpreprosteje je izbrati plast, ki predstavlja vhodni video, nato pa izbrati željeni učinek iz menija „Effects“ (slo. *Učinki*). S tem dodamo učinek na video. V oknu „Effect controls“ se s tem prikaže uporabniški vmesnik vtičnika. Vsak vtičnik ima lasten uporabniški vmesnik. Večinoma so sestavljeni iz vgrajenih klasičnih komponent, kot so drsniki, pojavna okna, gumbi in podobno, zato jih uporabniki AE z lahkoto razumejo, saj takšne kontrolnike že poznajo iz programa gostitelja.

Na posamezno plast lahko dodamo več učinkov, pri čimer je izhod iz enega uporabljen kot vhod naslednjega. Na ta način je omogočeno sestavljanje kompleksnih novih učinkov. Pri oblikovanju slike je dobro orodje za minimizacijo kompleksnosti predkompozicija (ang. *Precomposing*). S tem orodjem lahko eno ali več plasti izvozimo v novo kompozicijo. Predkomponirane plasti lahko

[2]Popravljanje barv (ang. *color correction* ali *color grading*) je dvodelen proces spreminjanja barv videa. V prvem koraku se popravi morebitne napake snemalca, kot so nepravilno izpostavljena slika, napačno izbrana nevtralna belina in podobno. Drugi korak je umetniški proces, pri katerem barvni tehnik s pomočjo režiserja prilagodi barve, da z njimi čustveno vpliva na gledalce. Primer močno in stilizirano popravljenih barv je film Matrica (ang. *The Matrix*, 1999). V filmu se s pomočjo močne zelene obarvanosti loči virtualno resničnost matrice od bolj nevtralno obarvanih prozorov, ki se dogajajo v resničnem svetu.

uporabljamo enako kot enostavne plasti, nanje dodajamo nove učinke in jih naprej komponiramo po želji. Predkompozicije lahko urejamo neodvisno od celotne kompozicije. Če se predkompozicija spremeni, je seveda potrebno ponovno upodobiti del kompozicije, na katerega ta sprememba vpliva.

Omogočena je tudi uporaba učinkov nad več plastmi hkrati s pomočjo posebnih prilagoditvenih plasti (ang. *Adjustment layer*). Če na prilagoditveno plast dodamo učinek, bo deloval nad vsemi učinki, ki so v skladu plasti pod njo. To omogoča enostavno in nedestruktivno urejanje celotne slike, kar je zelo priročno v določenih procesih post produkcije, naprimer pri popravljanju barv.

After Effects se pogosto uporablja v kombinaciji z Adobe Premiere Pro[3]. Med programskima paketoma je omogočena enostavna komunikacija. Montažer lahko iz Premiere Pro izvozi sceno ali kader naravnost v AE in jo nadomesti s kompozicijo. Ko v AE shranimo, se kompozicija v Premiere Pro posodobi in montažer lahko uporablja celoten kader z vizualnimi učinki.

[3]Adobe Premiere Pro je programski paket, namenjen predvsem video montaži. Sicer se zmogljivosti AE in Premiere Pro močno prekrivajo, vendar je vseeno mnogo lažje uporabljati vsak paket za njemu lasten namen.

Poglavje 4

Razvoj vtičnikov za After Effects

Za razvoj vtičnikov potrebujemo razvojno okolje za C++, SDK, ki ga ponuja Adobe na svoji spletni strani, in seveda programski paket After Effects. SDK vsebuje navodila za razvoj, opis delovanja vtičnikov v okolju programa After Effects in seveda opis funkcij, podatkovnih struktur in tipov, ki se uporabljajo v tem okolju. Polet tega je v SDK vključenih tudi veliko primerov vtičnikov. Vidimo lahko njihovo programsko kodo, jo spreminjamo ali uporabimo kot izhodišče za lastne vtičnike.

Adobe močno priporoča razvojno okolje Microsoft Visual Studio, če uporabljamo operacijski sistem Windows in Apple Xcode na operacijskem sistemu Max OS X. Poskrbeti moramo tudi, da imamo nameščene primerne verzije vseh orodij. Ker pri programiranju vtičnikov izhajamo iz primerov ali predlog, lahko hitro naletimo na težave, če imamo napačno verzijo razvojnega okolja ali SDK, ki ni namenjen razvoju za tisto verzijo AE, ki jo imamo nameščeno za testiranje.

Sam sem se odločil za delo na operacijskem sistemu MS Windows 10 Enterprise z okoljem MS Visual Studio 2015 update 3 za Adobe After Effects CC 2017.1 in Adobe After Effects CC 2017.1 Win SDK.

4.1 Delovanje vtičnikov

Vtičniki so knjižnjice DLL, ki jih After Effects med zagonom naloži. Nato poteka komunikacija med vtičnikom in programom gostiteljem tako, da se vtičnik odziva na klice gostitelja s primerno funckijo. Vtičnik ima vedno le eno vhodno točko, funckijo s podpisom:

```
PF_Err plugin_name (
    PF_Cmd cmd,
    PF_InData *in_data,
    PF_OutData *out_data,
    PF_LayerDef *output,
    void *extra)
```

After Effects ugotovi, kje se nahaja vhodna točka vtičnika s pomočjo resusra PiPL (ang. *plug-in property list* - seznam lastnosti vtičnika). To je dokument, ki ga napiše programer in katerega vsebina se mora ujemati z implementacijo vtičnika. Moderne verzije AE sicer večino lastnosti pridobijo direktno iz programske kode vtičnike in ne iz datoteke PiPL. Kljub temu se morajo podatki ujemati iz zgodovinskih razlogov in združljivosti s starejšimi verzijami datotek in vtičnikov.

Ko je vhodna funkcija poklicana, se mora na podlagi ukaznega izbirnika `cmd` pravilno odzvati. Vtičnik mora nekatere klice nujno podpreti, nekateri pa so opcijski. Vhodna točka je tako funkcija, ki skrbi le za to, da se bo prava funkcija odzvala na klic programa in pravilno reagirala v primeru napak. Vrne številko napake in poskusi brez neželjenih stranskih učinkov zaključiti izvajanje vtičnika.

Adobe priporoča, da kadar je le možno, uporabimo vgrajene funkcije in programske konstrukte, ki jih implementira AE in lahko do njih dostopamo prek SDK. S tem ne le skrajšamo čas, potreben za razvoj vtičnikov, ampak tudi zmanjšamo verjetnost za vpeljevanje hroščev in napak. Vgrajene

[1]Sekvenca v AE predstavlja samostojen projekt z enim ali več sloji videa in zvoka, nad katerimi izvajamo operacije.

ukazni izbirnik	**odziv**
Ukazni izbirniki, na katere se je potrebno odzvati ne glede na tip vtičnika	
PF_Cmd_ABOUT	Prikaži podatke o vtičniku - gostitelj bo pripravil dialog, vtičnik pa mora pripraviti vsebino.
PF_Cmd_GLOBAL_SETUP	Pripravi globalne podatke o vtičniku in rezerviraj pomnilnik za podatke, ki morajo biti dostopni ves čas delovanja vtičnika.
PF_Cmd_GLOBAL_SETDOWN	Če je v PF_Cmd_GLOBAL_SETUP bil rezerviran pomnilnik, ga sprosti. Sicer se ni potrebno odzvati.
PF_Cmd_PARAM_SETUP	Pripravi uporabniški vmesnik za podajanje parametrov vtičniku.
Za upravljanje s podatki sekvence [1]	
PF_Cmd_SEQUENCE_SETUP	Rezervira pomnilnik za podatke, ki niso odvisni od sličice, ki se obdeluje, ampak morajo biti dostopni tekom cele sekvence.
PF_Cmd_SEQUENCE_SETDOWN	Sprosti ves spomin, ki je bil rezerviran v PF_Cmd_SEQUENCE_SETUP
Ukazni izbirniki, ki se pošljejo za vsako sličico videa	
PF_Cmd_RENDER	Upodobi sličico. Klic zahteva, da vtičnik pripravi in v dodeljeni spomin zapiše vrednosti vseh pikslov. Klic se uporablja samo za prikazovanje slojev z barvno globino 8 ali 16 bitov na kanal.
PF_Cmd_SMART_PRE_RENDER	Se uporablja pri efektih tipa SmartFX. Vtičnik mora glede na željene pogoje prepoznati dele slike, ki jih bo upodobil.
PF_Cmd_SMART_PRE_RENDER	Klic SmartFX za upodobitev slike. V vnaprej prepoznane ciljne dele slike vpiši nove piksle.

Tabela 4.1: Nekateri izmed pogosteje podprtih ukaznih selektorjev.

funkcije seveda ne bodo delovale, če jih kličemo nepravilno in imajo vgrajene mehanizme za preprečevanje puščanja pomnilnika in sesuvanja. Pomembno je, da tudi, če vtičnik ne deluje pravilno, ne sesuje gostitelja. Če se sesuje gostitelj, lahko namreč končni uporabnik izgubi podatke in s tem svoje delo.

Zato nam naprimer SDK ponudi funkcije za rezervacijo pomnilnika, s katerimi ne tekmujemo z gostiteljem in operacijskim sistemom za delovni pomnilnik, saj rezervacijo za nas opravi gostitelj. S tem se izognemo situaciji, ko bi vtičnik porabil toliko spomina, da bi ga zmanjkalo za AE. S tem bi se sesul AE in z njim seveda tudi vtičnik, ki je odvisen od izvajanja programa gostitelja.

4.1.1 SmartFX

Posebna kategorija so vtičniki SmartFX. Od navadnih efektov se razlikujejo po tem, da omogočajo dvosmerno komunikacijo z AE in podpirajo video z barvno globino 32bitov na kanal[3]. Dvosmerno komunikacijo se doseže z implementacijo dodatnega ukaznega izbirnika `PF_Cmd_SMART_PRE_RENDER`, ki AE sporoči, katere dele slike bo upodobil. Ker vtičnik lahko ugotovi, katere dele slike mora upodobiti, je lahko upodabljanje veliko hitrejše. Upodabljanje slike z barvno globino 32 bitov na kanal je sicer lahko zelo počasno, saj uporablja operacije s plavajočo vejico in težje izkorišča vektorske enote modernih procesorjev. Za gladko uporabo AE in SmartFX potrebujemo podprto in zmogljivo grafično kartico, da se izognemo artefaktom in počasnemu delovanju[6].

4.2 Metoda razvoja

Pred razvojem željenega vtičnika se je dobro spoznati z uporabo SDK. V navodilih priporočajo, da si programer med primeri prebere in analizira tiste, ki so po načinu delovanja podobni ciljnemu[3]. Tega lahko nato začnemo postopoma spreminjati in testirati, da spoznamo, kako vtičnik komuniciira in sodeluje z aplikacijo AE. Marsikateri podatki v navodilih manjkajo ali pa

so zastareli, zato je potrebno kar veliko preizkušanja in raziskovanja. Skozi proces spoznavanja interakcije med programom in vtičnikom lahko oblikujemo strukturo vtičnika. Odločiti se moramo, kako razporediti operacije med funkcije in kako strukturirati podatke, da bodo dostopni ob pravem času in ne bodo ostajali v spominu. Glavno pri razvoju vtičnikov je pravzaprav, da se program pravilno odzove na vsak klic gostitelja posebej, hkrati pa ostane vtičnik logična enota. Seveda je prav tako pomembno, da je vtičnik uporaben in prijeten za uporabo z vidika končnega uporabnika. Na to moramo paziti pri oblikovanju uporabniškega vmesnika in tudi pri implementaciji učinka, da ostane odziven v različnih primerih uporabe. Med implementacijo si seveda lahko veliko pomagamo s razhroščevalnikom razvojnega okolja, navodili in Adobejevim forumom za razvijalce[2].

Implementacijo začnemo z ravojem prototipa, ki bo pravilno deloval na preprostih primerih. Na tej točki ni potrebno podpreti videa z večjo barvno globino, prav tako se ni potrebno ukvarjati s formati, katerih piksli niso kvadratni. Prototip razvijamo, dokler nismo zadovoljni z izgledom vizualnega učinka na preprostih primerih.

Naslednji korak je odpravljanje hroščev. Največ težav se je pojavilo z uporabo pomnilnika. Adobe priporoča uporabo funkcij za dostop do pomnilnika, ki jih ponuja SDK. Po mojih lastnih izkušnjah se je tega nasveta zelo pametno držati. V nasprotnem primeru imamo hitro lahko veliko puščanja pomnilnika. Vsaka sličica videa zavzame relativno veliko pomnilnika, saj mora biti med izvajanjem vtičnika shranjena nestisnjeno. Če nam ostane v pomnilniku, ki ga nismo pravilno sprostili in smo uporabljali sistemske klice za rezervacijo pomnilnika, se bo program AE sesul že po nekaj obdelanih sličicah. Do tega pride, ker AE poskusi alocirati pomnilnik, pa mu ga operacijski sistem ne more dodeliti. Če uporabljamo funkcije, ki jih ponuja SDK, se to ne more zgoditi, saj bo AE raje sesul vtičnik. S tem se izognemo izgubi podatkov - uporabnik lahko datoteko shrani in problematične vtičnike izklopi. Z vidika končnega uporabnika je veliko bolje, da se sesuje samo

[2]http://forums.adobe.com/community/aftereffects_general_discussion/aftereffects_sdk

vtičnik, kot pa celoten program.

Stabilne vtičnike, ki delujejo kot željeno, nato testiramo na robnih primerih. Ti vključujejo formate s širšimi piksli, večjo barvno globino (če to želimo podpreti), zelo velikim ali majhnim številom sličic na sekundo in podobno. Odpravljanje težav na tej stopnji je lahko zamudno in zapleteno, sploh če pred tem nismo odpravili bolj pogostih težav. Vprašati se je potrebno tudi, kakšne formate videa sploh želimo podpreti. Marsikateri vizualni učinek namreč ne bo pridobil ničesar, če ga izvajamo z barvno globino 32 bitov na kanal namesto s 16 biti ali celo 8 biti. Če razvijamo učinek za enkratno uporabo in ga ne nameravamo ponuditi javnosti, je dovolj dobro, da pravilno deluje na tipu videa, ki ga projekt uporablja.

4.3 Vtičnik PixelRain

4.3.1 Razvoj ideje

Prvi vtičnik, za katerega sem dobil idejo, sem poimenoval PixelRain. Ideja je zelo preprosta: nekatere piksle bo vtičnik prekopiral navzdol, s čimer se bodo na sliki pojavile črte. Kateri piksli se bodo kopirali in kateri ne, določi uporabnik na podlagi izbire barve, kateri podobne piksle bo vtičnik prekopiral. Poleg izbire barve bo uporabnik lahko določil tudi maksimalno dolžino sledi - kolikokrat se lahko piksel prekopira navzdol, in maksimalno razdaljo med percepcijo izbrane barve in barve piksla, da bo še štel kot podoben.

V originalni ideji smo podobnost barve definirali kot razdaljo med dvema barvama v prilagojenem prostoru RGB. Ker je cilj poiskati podobne barve, smo prostor RGB transformirali oziroma raztegnili, da bi razdalja bila bližja človeški percepciji razlike v barvah. Transformacija RGB prostora izhaja iz tega, da človeško oko naprimer zazna več različnih odtenkov zelene barve kot ostalih - bolj natančno ločimo zelene odtenke kot ostale. To upoštevamo v izračunu vizualne podobnosti barv tako, da razdaljo izračunamo po formuli[9]

$$\sqrt{2R^2 + 4G^2 + 3B^2}.$$

Formula je hiter približek, ki pa deluje dovolj natančno, da človek ne opazi razlike od bolj pravilno deformiranega barvnega prostora. Testiranje je to metodo pokazalo kot nerazumljivo in navidez nelogično. Pogosto so bili rezultati drugačni, kot bi jih pričakovali, kar je z vidika uporabnika nezaželjeno[14].

Po premisleku smo se odločili za implementacijo izbire barve s pomočjo treh parametrov v barvnem prostoru HSL[5]. Tako lahko uporabnik ločeno izbira barvni odtenek, nasičenost in svetllost. Ta način izbire barve sicer ne izbere barv, ki so enako oddaljene od ciljne v nobenem barvnem prostoru, je pa razumljiv in producira rezultate, kakršne uporabnik pričakuje. Zaradi ločenja na tri izbirnike smo s tem dosegli tudi večjo fleksibilnost vtičnika.

4.3.2 Implementacija vtičnika

Vtičnik implementira odzive na ukazne izbirnike `PF_Cmd_ABOUT`, `PF_Cmd_GLOBAL_SETUP`, `PF_Cmd_PARAMS_SETUP` in `PF_Cmd_RENDER`. To je minimalen nabor odzivov za efektni vtičnik.

Odziv na klic `PF_Cmd_ABOUT`

Funkcija `About()` je klicana, kadar uporabnik pritisne na gumb „About“ v oknu za kontrolo učinka.

```
static PF_Err
About (
    PF_InData *in_data,
    PF_OutData *out_data,
    PF_ParamDef *params[],
    PF_LayerDef *output )
{
    AEGP_SuiteHandler suites(in_data->pica_basicP);

    suites.ANSICallbacksSuite1()->sprintf(
    out_data->return_msg,
```

```
        "%s v%d.%d\r%s",
        STR(StrID_Name),
        MAJOR_VERSION,
        MINOR_VERSION,
        STR(StrID_Description));

    return PF_Err_NONE;
}
```

Vse funkcije, ki implementirajo odziv na ukazni izbirnik, so statične. Funkcija potrebuje le dva parametra, vednar so ostali tam zaradi večje berljivosti[3]. Ti štirje parametri so namreč prisotni v večini odzivnih funkcij, ki sestavljajo skelet vtičnika, tako da so hitro prepoznavne. Podporne funkcije, ki jih kličemo iz odzivnih funkcij, teh parametrov ponavadi nimajo. Parametri so:

- `PF_InData *in_data` je kazalec na strukturo, ki vsebuje podatke o stanju gostitelja in podatke, nad katerimi se vtičnik izvaja. Vsebina je odvisna tudi od tega, kateri ukazni izbirnik je poklican, saj je od tega odvisno, katere podatke bo vtičnik potreboval. Poleg tega lahko prek strukture dostopamo do kazalcev na funkcije, ki jih ponuja SDK. V tej funckiji uporabljamo le ta kazelec (`in_data->pica_basicP`).

- `PF_OutData *out_data` je struktura, ki vsebuje polja za vračanje podatkov iz vtičnika v AE. Ker se ista struktura uporablja za različne odzive vtičnika, so različna polja veljavna ob različnih časih in klicih. Tukaj uporabljamo polje `return_msg`, v katerega zapišemo tekst, ki se bo pojavil v pogovornem oknu. Za enostavnejšo lokalizacijo so nizi zapisani v drugi datoteki in se do njih dostopa prek globalne strukture s pomočjo makra `STR( ime_niza )`.

- `PF_ParamDef *params[]` je kazalec na seznam parametrov. V tej funkciji ni veljaven in se ne uporablja.

- `PF_LayerDef *output` je kazalec na izhod učinka. `PF_LayerDef` je struktura, ki vsebuje sliko kot seznam pikslov in metapodatke. Vanjo zapisušemo izhod upodobitve učinka. Tukaj kazalec ni veljaven in se ne uporablja.

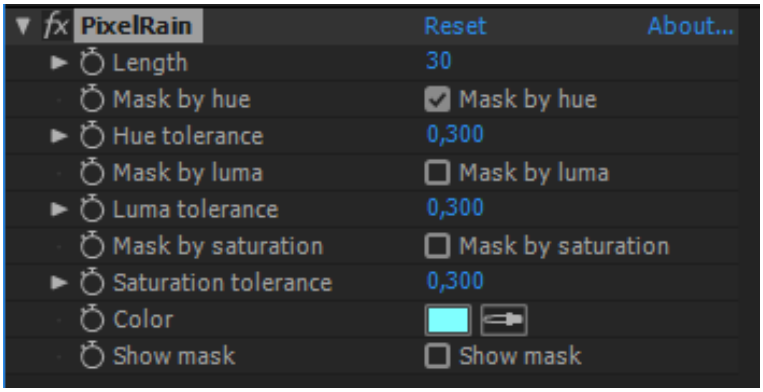
Funkcija `About` v vseh vtičnikih izgleda približno enako, saj je njem namen le to, da napolni dialog o vtičniku z vsebino. Vrne številko napake. V tej funkciji do napake tekom izvajanja ne more priti, razen če so težave prisotne v gostitelju, zato vedno vrne `PF_Err_NONE`.

Odziv na klic `PF_Cmd_GLOBAL_SETUP`

Funkcija `GlobalSetup` pripravi globalne podatke o vtičniku. To so različne zastavice, ki gostitelju povejo, kako vtičnik deluje in kaj zahteva od gostitelja. Funkcija tudi po potrebi rezervira spomin, ki bo vtičniku na voljo ves čas od njegove prve uporabe naprej. Ta funkcionalnost tukaj ni potrebna.

```
static PF_Err
GlobalSetup (
    PF_InData *in_data,
    PF_OutData *out_data,
    PF_ParamDef *params[],
    PF_LayerDef *output )
{
    out_data->my_version = PF_VERSION(
        MAJOR_VERSION,
        MINOR_VERSION,
        BUG_VERSION,
        STAGE_VERSION,
        BUILD_VERSION);
    out_data->out_flags = PF_OutFlag_NONE;

    return PF_Err_NONE;
```



Slika 4.1: Uporabniški vmesnik za nastavljanje parametrov.

```
}
```

Funkcija nastavi vrednosti verzije vtičnika in izhodne zastavice nastavi na `PF_OutFlag_NONE`, kar pomeni, da je vtičnik enostaven in podpira le 8 bitov na kanal barvne globine. Parametri so enaki kot pri funkciji `About`. V tej funkciji sicer ne potrebujemo niti kazalca na vgrajeno funkcionalnost SDK.

Odziv na klic `PF_Cmd_PARAMS_SETUP`

Funkcija `ParamsSetup` pripravi uporabniški vmesnik. Pripravljenih je kup makrov, ki poenostavijo dodajanje elementov uporabniškega vmesnika. Ko so vsi parametri dodani in pravilno nastavljeni, moramo gostitelju še sporočiti, koliko jih je, da ne pride do kakšne napake. Adobe priporoča, da se parametre dodaja tako, da ustvarimo en objekt tipa `PF_ParamDef` in ga za vsak parameter nastavimo, dodamo in spraznimo. Ker tip `PF_ParamDef` vsebuje unije, ki lahko vsebujejo podatke različnih tipov glede na tip parametra, je zelo pomembno, da so parametri izpraznjeni. V ta namen je na voljo makro `AEFX_CLR_STRUCT`. Makro prepiše ves pomnilnik, ki ga struktura zasede, z ničlami.

```
static PF_Err
ParamsSetup (
```

```
    PF_InData       *in_data,
    PF_OutData      *out_data,
    PF_ParamDef     *params[],
    PF_LayerDef     *output )
{
    PF_Err      err     = PF_Err_NONE;
    PF_ParamDef def;

    AEFX_CLR_STRUCT(def);
    // length slider
    PF_ADD_FLOAT_SLIDERX(
        STR(StrID_Length_Param_Name),
        PIXELRAIN_LENGTH_MIN,
        PIXELRAIN_LENGTH_MAX,
        PIXELRAIN_LENGTH_MIN,
        PIXELRAIN_LENGTH_MAX,
        PIXELRAIN_LENGTH_DFLT,
        PF_Precision_INTEGER,
        0, 0,
        LENGTH_DISK_ID);
    AEFX_CLR_STRUCT(def);
    // hue cb
    PF_ADD_CHECKBOX(
        STR(StrID_HueCB_Param_Name),
        STR(StrID_HueCB_Param_Name),
        true,
        NULL,
        HUECB_DISK_ID);
    AEFX_CLR_STRUCT(def);
    // hue slider
    PF_ADD_FLOAT_SLIDERX(
```

```
        STR(StrID_HueSlider_Param_Name),
        0, 1, 0, 1, 0.3,
        PF_Precision_THOUSANDTHS,
        0, 0,
        HUETOL_DISK_ID);
    ...
    // color picker
    PF_ADD_COLOR(
        STR(StrID_Color_Param_Name),
        PF_HALF_CHAN8,
        PF_MAX_CHAN8,
        PF_MAX_CHAN8,
        COLOR_DISK_ID);
    AEFX_CLR_STRUCT(def);
    // mask only cb
    PF_ADD_CHECKBOX(
        STR(StrID_MaskCB_Param_Name),
        STR(StrID_MaskCB_Param_Name),
        false,
        NULL,
        MASKCB_DISK_ID);
    AEFX_CLR_STRUCT(def);

    out_data->num_params = PIXELRAIN_NUM_PARAMS;

    return err;
}
```

Funkcija vrača številko napake, ki jo nastavi neuspešno izveden makro.

Odziv na kliv `PF_Cmd_RENDER`

Funkcija `Render` implementira upodabljanje izhodne slike. To poteka v dveh korakih. Prvi korak je generiranje maske na podlagi izbrane barve. Drugi korak je generiranje sledov na podlagi maske in vhodne slike. Pred tem je seveda potrebno pridobiti vse potrebne podatke in pripraviti podatkovne strukture.

```
static PF_Err
Render (
        PF_InData *in_data,
        PF_OutData *out_data,
        PF_ParamDef *params[],
        PF_LayerDef *output )
{
    ...
```

Funkcija sprejme enake parametre kot prej opisane. Med klicem za upodobitev pa se `params` in `output` obnašata drugače kot pri ostalih funkcijah.

- `PF_ParamDef *params[]` je veljavna in kaže na seznam vseh parametrov. Prvi parameter `params[0]` je kazalec na vhodno sliko tipa `PF_LayerDef *`. Ostali so tipa `PF_ParamDef` in vsebujejo podatke o stanju kontrolerjev, ki sestavljajo uporabniški vmesnik.

- `PF_LayerDef *output` je izhodna slika, ki je enaka vhodni, vendar prazna - stanje pikslov ni znano. Vanjo zapišemo nove vrednosti pikslov.

```
    ...
    PixSelInfo psi;
    AEFX_CLR_STRUCT(psi);
    psi.hueCheck = params[PIXELRAIN_HUE_CB]->u.bd.value;
    psi.hueTolerance =
        params[PIXELRAIN_HUE_TOLERANCE]->u.fs_d.value;
    ...
```

Začetni del kode poskrbi za dostopnost kazalca na funkcije SDK in podatke iz elementov uporabniškega vmesnika prepiše v strukturo, ki bo podana funkciji za pripravo maske. Na primeru vidimo, kako iz parametrov preberemo podatke. Ker so parametri vsi istega tipa, vsebujejo pa različne podatke in predstavljajo različne elemente UI, so podatki zapisani v unijah. Ko dostopamo do podatkov, moramo poskrbeti, da podatek pravilno interpretiramo oziroma prepišemo pravi element unije.

```
/* 1. Generate pixel mask */
    PF_Handle colorMaskH =
        suites.HandleSuite1()->host_new_handle(sizeof(PF_EffectWorld));
    PF_EffectWorld * colorMaskP = (PF_EffectWorld*)*colorMaskH;
    ERR(suites.WorldSuite1()->new_world(NULL,
        output->width,
        output->height,
        NULL,
        colorMaskP));
    if (err != PF_Err_NONE) return err;

    ERR(suites.Iterate8Suite1()->iterate(in_data,
        0,
        linesL,
        &params[PIXELRAIN_INPUT]->u.ld,
        NULL,
        (void*)&psi,
        GenerateMaskPixFunc,
        colorMaskP));

    // show mask
    if ((PF_Boolean)params[PIXELRAIN_SHOW_MASK]->u.bd.value) {
        suites.WorldTransformSuite1()->copy(NULL,
            colorMaskP,
```

```
            output,
            NULL,
            NULL);
        return err;
    }
```

Naslednji del funkcije pripravi masko, katere piksli imajo vrednost alpha 1.0, če so dovolj podobni izbrani barvi - v mejah uporabniško določenih toleranc. Po vsakem parametru (odtenek, zasičenost in svetlost) maskiramo ločeno.

```
    ...
    /* 1. Hue mask */
    if (psiP->hueCheck) {
        double maxHueDiff = psiP->hueTolerance / 2.0;
            // in UI diff of 1.0 covers all, in code 0.5
        double hueTgt = tgtHSL->hue;
        double hueIn = inHSL->hue;
        double hueDiff = 0.5 - abs(abs(hueTgt - hueIn) - 0.5);
            // hue wraps around
        if (hueDiff > maxHueDiff) {
            // alpha is mask, color tells which property masked
            outP->alpha = maskAlpha;
            outP->red = 255; // red for hue mask
        }
    }
    ...
```

Najprej pripravi prazno sliko, v katero se bo maska zapisala. Ukaz `suites.HandleSuite1()->host_new_handle(...);` zahteva od gostitelja, naj mu varno rezervira željeno količino pomnilnika. Funkcija vrača ročico (kazalec na kazalec), ki jo shranimo, saj jo bomo potrebovali, da bo gostitelj lahko sprostil pomnilnik, ko ga ne bomo več potrebovali. Ta pomnilnik uporabimo,

da ustvarimo novo sliko tipa `PF_Effect_World`. Tipa `PF_Effect_World` in `PF_Layer_Def` sta enaka, vendar nekatere funkcije SDK zahtevajo parametre prvega tipa, nekatere pa drugega. Da se izognemo opozorilom prevajalnika, lahko ta dva tipa brez težav spreminjamo iz enega v drugega.

S pomočjo iteratorske funkcije ustvarimo masko. Iteratorska funkcija `suites.Iterate8Suite1()->iterate( ... )` za nas pokliče funkcijo, podano prek funkcijskega kazalca, ki se izvede nad vsakim pikslom vhodne slike. Uporaba iteratorskih funkcij je priporočena, saj so optimizirane za izvajanje na večjedrnih procesorjih. To nam prihrani nekaj dela za hitrejše izvajanje vtičnika.

Funkcija za obdelavo piksla `GenerateMaskPixFunc` kot najpomembnejši vhod prejme piksel. Piksli so shranjeni v formatu RGB, zato jih je za filtriranje po kanalih HSL potrebno pretvoriti, kar sem naredil po prilagojeni metodi s spletne strani[13]. AE SDK vključuje funkcije za takšne pretvorbe, vendar so namenjene pretvarjanju celotne slike z veliko natančnostjo. Implementacija lastne hitre in dovolj natančne funkcije je pospešila izvajanje upodabljanja.

Če je uporabnik izbral način prikazovanja maske (v UI ang. *Show mask*), z masko prepišemo izhodno sliko. Nadaljno delo ni potrebno, saj smo prikazali masko. Namen tega načina delovanja je, da uporabnik vidi, kateri deli slike bodo pustili sledi, in s tem lažje oblikuje vizualni učinek.

```
/* 2. Generate trails */
    ...
    PF_Handle trailsWorldH =
        suites.HandleSuite1()->host_new_handle(sizeof(PF_EffectWorld));
    PF_EffectWorld * trailsWorldP = (PF_EffectWorld*)*trailsWorldH;
    ERR(suites.WorldSuite1()->new_world(
        NULL,
        output->width,
        output->height,
        NULL,
```

```
        trailsWorldP
    ));
    if (err != PF_Err_NONE) return err;

    tilP->output = reinterpret_cast<PF_LayerDef*>(trailsWorldP);

    ERR(suites.Iterate8Suite1()->iterate_generic(
        output->width,
        (void*)tilP,
        GenerateTrailColFunc8
    ));
    if (err != PF_Err_NONE) return err;

    // show trails
    ERR(suites.WorldTransformSuite1()->copy(
        NULL,
        tilP->output,
        output,
        NULL,
        NULL
    ));

    /* Free memory */
    suites.HandleSuite1()->host_dispose_handle(colorMaskH);
    suites.HandleSuite1()->host_dispose_handle(trailsWorldH);

    return err;
}
```

Drugi del funkcije pripravi podatke, potrebne za prikaz sledi. Kot pri generiranju maske tudi tukaj prepišemo potrebne podatke v strukturo `til`. Zaradi preglednosti kode in težav z osveževanjem pomnilnika, ki jih ima

AE, vtičnik generira sledi v novo sliko. Novo sliko je potrebno ustvariti, ker sicer AE poskuša optimizirati delovanje in se lahko zgodi, da imajo v izhodni sliki prazni piksli vrednosti prejšnjih sličic videa. Uporabnikom AE SDK priporočamo, da v primeru artefaktov tipa *ghosting* (prikazovanje delov prejšnjih sličic, pogosto deformiranih) poskusijo ročno prepisati izhodno sliko. To je rešilo težave tega vtičnika.

Iteratorske funkcije za dostop do pikslov ne zagotavljajo nebenega definiranega zaporedja dostopov, zato jih zdaj ne moremo uporabiti. Algoritem za generiranje sledov je namreč enostaven in zahteva, da se vsak stolpec pikslov obdela po vrsti od zgoraj navzdol oziroma v smeri naraščajoče koordinate Y. Pomagamo si s splošnim iteratorjem, s katerim iteriramo po stolpcih. Za vsak stolpec se pokliče funkcijo `GenerateTrailColFunc8`. V funkciji nato ročno dostopamo do posameznih pikslov, pri čemer nam pomagajo makri za dostop do pikslov `PF_GET_PIXEL_DATA8`. Makro kazalec na piksel tipa `PF_Pixel8` usmeri na prvo mesto polja pikslov, katerih tipe prilagodi željeni barvni globini. S temi kazalci lahko potem počnemo vse pričakovano. Zaradi interne podatkovne strukture slike je potrebno paziti na dolžino posamezne vrstice. Ta namerč nima enake dolžine širini slike, je pa njena dolžina definirana v podatkovni strukturi `PF_Effect_World`. V novo vrstico istega stolpca se premaknemo s `inP += tilP->input->rowbytes / sizeof(PF_Pixel8);` , kjer je `inP` kazalec na piksel, `tilP->input` pa kazalec na sliko.

4.4 Vtičnik ElStAberr

4.4.1 Razvoj ideje

Osnovna ideja za tem vizualnim učinkom je simulacija elektro statičnega odklona (ang. *electro static aberration*)[**?**]. Gre za pojav, ki ga je potrebno aktivno negirati pri oblikovanju katodnih ekranov, naprimer televizij. Takšni ekrani vsebujejo elektronski top in elektromagnete, s katerimi se snop elektronov usmerja. Vsak piksel mora biti ob pravem času „osvetljen“ s primerno močnim elektronskim snopom, da florescentni premaz zažari s primerno sve-

Slika 4.2: Primer vhodne slike.

Slika 4.3: Primer slike z vizualnim učinkom, ki uporablja vtičnik PixelRain.

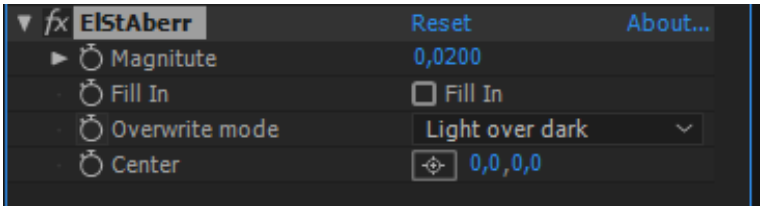
tlostjo. Eleketronski žarek z večjo energijo pa potrebuje več energije, da se ga preusmeri. Zato ekrani potrebujejo vezje, ki poveča moč elektromagnetov za preusmerjanje žarka, kadar ima žarek večjo energijo. Vizualni učinek ElStAberr približno simulira, kako bi izgledala slika, če bi bila prikazana na katodnem zaslonu brez takšne logike.

Pri barvnem katodnem ekranu je vsak piksel sestavljen iz treh delov - en zažari rdeče, en zeleno, en pa modro. Zato mora tudi elektronski snop zadeti te tri točke ločeno in z različno močjo. Zato je tudi možno, da se trije snopi različno napačno odklonijo. Ker pa mora snop zadeti pravo točko, je možno celo, da zadane del ekrana, ki predstavlja drugo barvo od ciljane. Vse to smo pri tej simulaciji zanemarili in vsak piksel premaknili glede na njegovo svetlost, ostale detajle pa zanemarili. Rezultat sicer ni fizikalno pravilna simulacija, je pa vizualno zanimivo orodje, s katerim lahko dosežemo različne zanimive vizualne učinke.

Namesto pravilne simulacije smo se usmerili k izdelavi uporabnega vtičnika. Dodali smo kontrolnik za izbiro središča vizualnega učinka. Predstavlja točko, v kateri je odklon vedno ničeln. Tej točki je v dejanskem ekranu analogen navidezni izvor elektronskega snopa oziroma točka, ki jo žarek zadane brez odklona. Dodali smo tudi možnost zapolnitve lukenj, ki nastanejo, kadar je piksel premaknjen s svojega mesta in na to mesto ni premaknjen noben drug piksel. Ta opcija nima analoga v pravem ekranu s katodno cevjo, vendar poveča fleksibilnost učinka in s tem njegovo uporabnost.

Uporabniški vmesnik vsebuje sledeče kontrole:

- moč odklona, ki je lahko pozitiven - svetli piksli so potisnjeni stran od sredine slike, temni pa proti sredini, ali pa negativen z obratnimi rezultati,
- zapolnitev lukenj - ker premikamo piksle, je možno, da nekateri piksli ostanejo prazni. S to opcijo se prazni piksli zapolnijo z najbližjim sosedom,
- način delovanja za prepisovanje pikslov, v katere se premakne več kot en vhodni piksel. Lahko damo prednost svetlejšim pikslom, da prepišejo temne, prednost temnih pred svetlimi, seštevanje obeh, kar je najbližje simulaciji, ali povprečje obeh pikslov,
- izbira sredinske točke odklona - izberemo, kje na sliki je središče vizualnega učinka - navidezni izvor elektronskega žarka.



Slika 4.4: Uporabniški vmesnik za nastavljanje parametrov.

4.4.2 Implementacija vtičnika

Vtičnik implementira odzive na ukazne izbirnike `PF_Cmd_ABOUT`, `PF_Cmd_GLOBAL_SETUP`, `PF_Cmd_PARAMS_SETUP`, `PF_Cmd_RENDER` in `PF_UPDATE_PARAMS_UI`. Odziva na ukazna izbirnika `PF_Cmd_ABOUT` in `PF_Cmd_GLOBAL_SETUP` sta enaka kot pri vtičniku PixelRain, le da funkcija `GlobalSetup` tudi opozori AE na to, da vtičnik implementira tudi odziv na ukazni izbirnik `PF_UPDATE_PARAMS_UI`.

Med upodabljanjem sličic se izračuna odklon na podlagi svetlosti piksla, nato pa se piksel premakne na novo mesto. Pri tem je možno, da se na isto mesto premakne več kot en piksel. To težavo rešujemo s prepisovanjem ali mešanjem obeh pikslov na tem mestu. Način delovanja izbere uporabnik.

Zaradi premikanja pikslov stran od centra učinka se v izhodni sličici lahko pojavijo prazna območja brez pikslov. Uporabnik lahko izbere, da se ta območja zapolnijo, kar pa deluje le za manjša območja. Prazna območja so velika, kadar je moč odklona velika. V takih primerih zapolnjevanje slike nima velikega smisla in ni podprto. Ugotavljanje, kateri piksel manjka oziroma s kakšnimi piksli napolniti praznine, s trenutno hitro implementacijo ni možno. Rešiti ta problem bi bilo zanimivo, najverjetneje bi bilo potrebno implementirati metanje žarkov v 3D prostoru.

Odziv na klic `PF_Cmd_PARAMS_SETUP`

Funkcija `ParamsSetup` je enaka kot pri vtičniku PixelRain. V polje parametrov doda paramtre, ki jih vtičnik uporablja. Omembe vreden je parameter za izbiro cetralne točke tipa `PF_Param_POINT`. Ko ga dodamo, mu seveda

lahko nastavimo prednastavljeno vrednost. Želimo jo nastaviti na sredino platna, kar bodo uporabniki pričakovali in najverjetneje pogosto uporabljali. Ker pa ob dodajanju parametrov še ne vemo, kako veliko je platno, tega ne moremo narediti. Lahko bi sklepali, da bo platno dandanes najpogosteje uporabljane velikosti HD (širina 1920 pikslov in višina 1080), vendar se na to ne moremo zanašati. Ne želimo, da bi v primeru majhnega platna središče učinka šlo izven platna. Temu problemu smo se izognili z implementacijo odziva na ukazni izbirnik `PF_Cmd_UPDATE_PARAMS_UI`. Da bo AE poslal ukazni izbirnik, zagotovimo z nastavitvijo zastavice `PF_ParamsFlag_SUPERVISE`.

```
...
def.flags = PF_ParamFlag_SUPERVISE;
PF_ADD_POINT(
    STR(StrID_Center_Param_Name),
    50,
    50,
    NULL,
    CENTER_DISK_ID);
AEFX_CLR_STRUCT(def);
...
```

Odziv na klic `PF_Cmd_UPDATE_PARAMS_UI`

Funkcija `UpdateParamaterUI` se pokliče, kadar mora AE spremeniti uporabniški vmesnik vtičnika. To so lahko kompleksne spremembe, kot je dodajanje parametrov ali spreminjanje tipov parametrov, ali pa preproste, kot je na primer spreminjanje vrednosti parametrov. Klic lahko tudi programsko sprožimo s pomočjo vgrajene funkcije.

Funkcijo `UpdateParamaterUI` uporabimo, da nastavimo vrednost parametra za izbiro centra učinka, ko je velikost platna že znana. Tako se izognemo neželjenim prednastavljenim vrednostim parametra.

```
// sets up default center point in the center
```

```
    // before this call, canvas size is unknown
    if (outputP) {
        def = *params[ElStAberr_CENTER];
        def.u.td.x_dephault = outputP->width / 2;
        def.u.td.y_dephault = outputP->height / 2;
        def.u.td.x_value = outputP->width / 2;
        def.u.td.y_value = outputP->height / 2;
        suites.ParamUtilsSuite3()->PF_UpdateParamUI(
            in_data->effect_ref,
            ElStAberr_CENTER,
            &def);
    }
```

Odziv na klic `PF_Cmd_RENDER`

Funkcija `Render` upodobi sličico korakoma. Najprej napolnimo celotno izhodno platno z barvo RGBA(1.0, 0.0, 1.0, 0.0). To so popolnoma prozorni piksli magenta barve. Ker so v AE piksli z alpha vrednostjo 0 vedno črne barve, nam bo to kasneje pomagalo zaznati nezapolnjene piksle.

```
static PF_Err
ColorBlankPreAberGapsPixFunc8(
    void        *refcon, // null
    A_long      xL,
    A_long      yL,
    PF_Pixel8   *inP, // null
    PF_Pixel8   *outP)
{
    outP->alpha = 0;
    outP->red = 255;
    outP->green = 0;
    outP->blue = 255;
    return PF_Err_NONE;
```

```
    ...
```

Naslednji korak je ključen. Vsakemu pikslu se izračuna njegova svetlost in na njeni podlagi tudi moč odklona za piksel. Pri tem se seveda upošteva parameter moči učinka. Izračuna se nova lokacija piksla na podlagi njegovih koordinat, moči odklona in lokacije centra učinka. Nato se piksel zapiše na novo lokacijo v izhodno platno, ki je bilo pred tem označeno kot prazno. Pred zapisovanjem se preveri, če je na istih koordinatah že postavljen piksel. Če je, se na podlagi izbranega načina prepisovanja ali mešanja primerno izračuna končna barva piksla.

Če uporabnik ne želi polniti praznih območij, se upodobljeno sliko prepiše na izhod.

```
static PF_Err
LumaAberratePixFunc8 (
    void        *refcon,
    A_long      xL,
    A_long      yL,
    PF_Pixel8   *inP,
    PF_Pixel8   *outP) // output unused
{
    ...
    double luma = lumaFromColor(inP); // luma [0.0, 1.0]
    // remember state
    int oldX, oldY;
    // recenter image
    int centerX, centerY;
    centerX = (int)aiP->center_x;
    centerY = (int)aiP->center_y;

    oldX = xL - centerX;
    oldY = yL - centerY;
```

```
    // calculate scaling factor
    int newX, newY;
    double multiplier = aiP->magnitudeF * 2.0; // [-2, 2]
    double modifier = aiP->magnitudeF * -1.0 + 1.0; // [+2, 0]
    double scaleDelta = multiplier * luma + modifier;
    // scale image
    newX = (int)(oldX * scaleDelta);
    newY = (int)(oldY * scaleDelta);

    // recenter image
    newX += centerX;
    newY += centerY;
    ...
```

Če je izbran način polnjenja praznin, se za vsak piksel preveri, če je prazen. Brez težav lahko ločimo nezapisane piksle, ki so bili na začetku označeni s prosojno magenta barvo. Če je piksel prazen, se poišče najbližji zapisan piksel v smeri proti centru učinka in prepiše na novo mesto. Pri tem iščemo le nekaj pikslov daleč, saj velikih lukenj ne polnimo.

```
static PF_Err
Render (
    PF_InData       *in_data,
    PF_OutData      *out_data,
    PF_ParamDef     *params[],
    PF_LayerDef     *output )
{
    ...
    // 3. Fill emptynesses
    if (params[ElStAberr_FILLIN]->u.bd.value) { // is checked
        ...
```

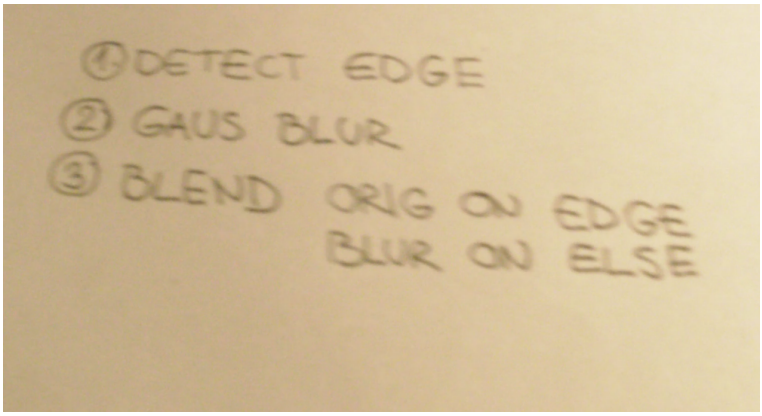
```
        // transfer mode for fill in
        PF_Handle simpleBehindH =
            (PF_Handle)suites.HandleSuite1()->host_new_handle(
                    sizeof(PF_CompositeMode));
        PF_CompositeMode *simpleBehind = (PF_CompositeMode*)*simpleBehindH;
        simpleBehind->opacity = 100;
        simpleBehind->xfer = PF_Xfer_BEHIND;
        // blend fill in under output
        suites.WorldTransformSuite1()->composite_rect(
            in_data->effect_ref,
            NULL,
            INT_MAX, // A_long opacitity -- 0 to int32 max
            fillInWorldP,
            0, 0,
            PF_Field_FRAME,
            PF_Xfer_BEHIND,
            outworldP
        );
    ...
```

Slika 4.5: Primer vhodne slike.

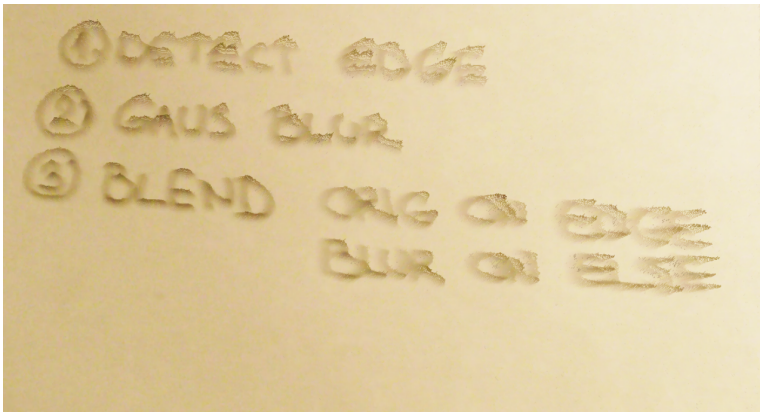
Slika 4.6: Primer slike z vizualnim učinkom, ki uporablja vtičnik ElStAberr.

Slika 4.7: Primer vhodne slike.

Slika 4.8: Primer preoblikovanja vizualnih elementov z vtičnikom ElStAberr.



Slika 4.9: Primer vhodne slike.

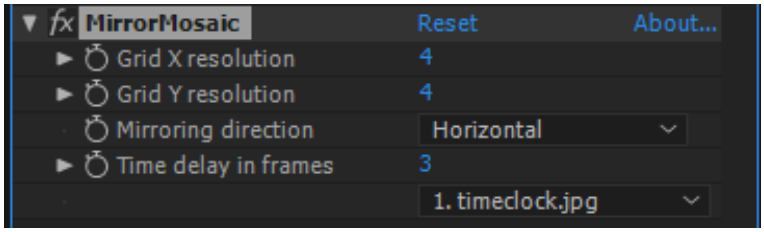


Slika 4.10: Primer slike z dodano globino oziroma učinkom reliefnega tiska, narejene s pomočjo vtičnika ElStAberr.

4.5 Vtičnik MirrorMosaic

4.5.1 Razvoj ideje

Idejo za vtičnik je prispeval umetnik Vanja Mervič. Gre za digitalni vizualni učinek, ki sestavi video iz več kopij dela videa. Vsak naslednji video je za nekaj sličic zamaknjen v času in prezrcaljen. Rezultat je delno simetričen



Slika 4.11: Uporabniški vmesnik za nastavljanje parametrov.

video mozaik.

4.5.2 Implementacija vtičnika

Vtičnik MirrorMosaic je strukturno enak vtičniku PixelRain. Vtičnik deluje tako, da na podlagi uporabniško izbranih parametrov najprej sestavi sliko iz delov, nato pa prezrcali željene dele slike. Vtičnik mora za implementacijo časovnega zamika dostopati do sličic videa ob različnih časih. V AE je vsaka sličica videa iz vhodne datoteke dekodirana, kadar se jo potrebuje, saj nestisnjen video zavzame veliko preveč spomina, da bi lahko bil v celoti naložen. Do sličic videa ob drugih časih, kot je truntni[3], dostopamo s temu namenjenimi funkcijami. Pomembno je, da pazljivo delamo s pomnilnikom, ki se lahko hitro zapolni in sesuje vtičnik ali celo program AE.

Odziv na ukazni izbirnik `PF_Cmd_GLOBAL_SETUP`

Funkcija `GlobalSetup` je enaka kot pri drugih vtičnikih, s to razliko, da mora vtičnik opozoriti gostitelja na to, da dostopa do sličic v „ne-trenutnem času“. Če bi vtičnik podprl barve z globino 32bit na kanal v plavajoči vejici, bi za povečanje hitrosti delovanja nastavil še zastavico, ki AE prisili v predčasno nalaganje sličic. Za video manjše barvne globine to ni podprto.

```
...
// using frames from other times, not just current_time
```

[3]Trenutni čas je čas sličice, ki jo obdelujemo.

```
out_data->out_flags = PF_OutFlag_WIDE_TIME_INPUT;
// for 32bpc float colors, unsuported
//out_data->out_flags2 = PF_OutFlag2_AUTO_WIDE_TIME;
...
```

Odziv na ukazni izbirnik PF_Cmd_RENDER

Funkcija Render na podlagi vhodnih parametrov pripravi pravokotnik, ki predstavlja eno kopija videa v končnem mozaiku. Velikost se preprosto izračuna na podlagi željenega števila horizontalnih in vertikalnih ponovitev.

```
...
/* 1. Grid setup */
int gridX, gridY;
gridX = (int)params[MIRR_GRIDX]->u.fs_d.value;
gridY = (int)params[MIRR_GRIDY]->u.fs_d.value;
int dX = output->width / gridX; // width of rect
int dY = output->height / gridY; // height of rect
PF_Rect in_rect = { 0, 0, dX, dY }; // rect coords type: A_long
...
```

Naslednji korak je kopiranje vhodnega videa po platnu. To opravimo v zanki, v kateri s pomočjo makra PF_CHECKOUT_PARAM dostopamo do vhodnega videa ob poljubnem času. Za vsako kopijo izračunamo koordinate izhodnega območja in relativen zamik v času. V ciljno območje prepišemo sličico. Če je časovni zamik večji kot preostala dolžina videa, pustimo območje prazno. S tem omogočimo uporabo učinka za generiranje neopazno ponavljajočega se videa v AE. Po zaključku izvajanja zanke je video sestavljen kot mozaik časovno zamaknjenih kopij sebe.

```
/* 2. LOOP: checkout frame, place and mirror */
PF_ParamDef checkout; // layer from time+dT goes here
AEFX_CLR_STRUCT(checkout);
```

```
int dT = (int)params[MIRR_DT]->u.fs_d.value;
int iX, iY;
for (iX = 0; iX < gridX; iX++) {
    for (iY = 0; iY < gridY; iY++) {
        int deltaT = 0;
        if (params[MIRR_MIRRDIR]->u.pd.value == MIRR_HOR) {
            deltaT = dT * iX;
        }
        else {
            deltaT = dT * iY;
        }
        // checkout from time
        AEFX_CLR_STRUCT(checkout);
        ERR(PF_CHECKOUT_PARAM(
            in_data,
            MIRR_LAYER,
            (in_data->current_time + deltaT * in_data->time_step),
            in_data->time_step,
            in_data->time_scale,
            &checkout));

        if (!err) {
            // prepare dest rect
            dest_rect.left = iX * dX;
            dest_rect.right = (iX + 1) * dX;
            dest_rect.top = iY * dY;
            dest_rect.bottom = (iY + 1) * dY;

            // place rect if frame exists
            if (checkout.u.ld.data) {
```

```
                    // first copy, then mirror
                    ERR(suites.WorldTransformSuite1()->copy(
                        in_data->effect_ref,
                        &checkout.u.ld,
                        output,
                        &in_rect,
                        &dest_rect));

                }
                // if frame not exist, empty black instead
                else {
                    ERR(PF_FILL(NULL, &dest_rect, output));
                      // NULL = RGBA(0,0,0,0)
                    continue;
                }
            }
            // every checkout must be checked-in
            PF_CHECKIN_PARAM(in_data, &checkout);
        }
    }
```

Naslednji korak, zrcaljenje, je implementiran z dvema funkcijama za piksle, `MirrorHorPixFunc8(...)` in `MirrorHorPixFunc8(...)`. Razlog za ločeno implementacijo horizontalnega in vertikalnega zrcaljenja je večja enostavnost kode. Tako je tudi nadgradnja učinka, da bi omogačal zrcaljenje v obe smeri hkrati, preprosta.

Delovanje obeh funkcij je podobno. Za vhodni piksel se na podlagi njegovih koordinat izračuna, če ga je potrebno zrcaliti ali ne. Preveri se, v katerem pravokotnem območju se nahaja. V tem območju se nato izračunajo njegove relativne koordinate in nove koordinate po zrcaljenju. Piksel se nato prepiše v izhodno območje. Da ne pride do napak tipa pisanje pred branjem ali pisanje pred pisanjem, morata biti vhodno in izhodno platno različni. To sicer

porabi nekaj dragocenega pomnilniškega prostora, ki pa ga kmalu sprostimo.

```
static PF_Err
MirrorHorPixFunc8(
    void        *refcon, // MirrorInfo
    A_long      xL,
    A_long      yL,
    PF_Pixel8   *inP, // unused
    PF_Pixel8   *outP) // only outP is writable
{
    PF_Err      err = PF_Err_NONE;
    MirrorInfo *miP = (MirrorInfo*)refcon;
    PF_InData *in_data = miP->in_data;
    PF_Pixel8 *inpixP = NULL;
    PF_LayerDef *input = (PF_LayerDef*)miP->input;
    int newX = xL;
    int newY = yL;

    int rectWidth = miP->rect->right - miP->rect->left;
    int rectNoX = xL / rectWidth; // starting at 0
    int rectHeight = miP->rect->bottom - miP->rect->top;
    int rectNoY = yL / rectHeight; // starting at 0

    if (rectNoX % 2 == 1) {
        int relX = xL % rectWidth;
        newX = xL + rectWidth - 2 * relX;
    }

    ERR(PF_GET_PIXEL_DATA8(miP->input, NULL, &inpixP));
    if (!err) {
        inpixP += newY * miP->input->rowbytes / sizeof(PF_Pixel8) + newX;
        outP->alpha = inpixP->alpha;
```

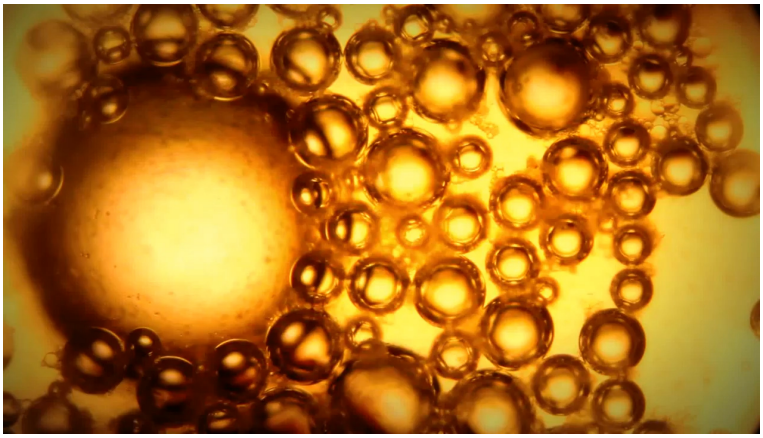
```
        outP->red = inpixP->red;
        outP->green = inpixP->green;
        outP->blue = inpixP->blue;
    }

    return err;
}
```

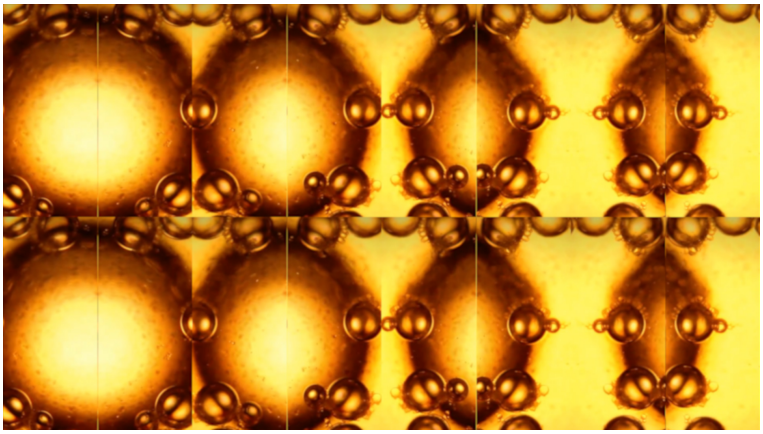
Vtičniku bi bilo za lepši rezultat treba dodati nekaj funkcij, kot so mehčanje prehodov med posameznimi kopijami videa in bolj zanesljivo zrcaljenje - trenutna implementacija za nekatere nastavitve parametrov ne producira pravilnih rezultatov.

Slika 4.12: Primer vhodne slike.

Slika 4.13: Primer slike, ustvarjene z vtičnikom MirrorMosaic brez časovnega zamika.



Slika 4.14: Primer vhodne slike.



Slika 4.15: Primer slike, ustvarjene z vtičnikom MirrorMosaic s časovnim zamikom.

Poglavje 5

Zaključek in ugotovitve

Tekom dela smo opravili pregled zgodovinskega razvoja vizualnih učinkov skozi razvoj različnih tehnologij. Novi vizualni učinki so se ponavadi razvijali na osnovi starejših, po namenu podobnih. Nove tehnologije so skozi zgodovino prinašale nove metode dela z vizualnimi učinki. Na primeru razlike med uporabo vizualnih učinkov v filmu in videu je očitno, kako se tehnološka razlika pozna tudi v umetniški uporabi medija. Film, katerega uporaba je zaradi tehnoloških omejitev draga in časovno potratna, se uporablja za pripovedovanje zgodb. Skozi leta se je razvijala kvaliteta slike, zvoka, filmski jezik se je razvijal in vizualni učinki so postajali bolj in bolj prepričljivi. Video se je zaradi lastnosti medija in tehnoloških možnosti in omejitev razvijal v drugo smer. Uporabljal se je za izdelavo avtorskih umetniških izdelkov, ki so raziskovali meje tehnologije in umetniškega izražanja skozi eksperimentiranje. Pojav in hiter razvoj digitalnega videa je oba pristopa približal. Danes se digitalni video uporablja v filmskem svetu in kot video v umetniških produkcijah. Razvil je tudi lasten izrazni jezik, ki pa je marsikaj prevzel iz filmskega jezika in tudi videa.

Razvili smo tri vtičnike za vizualne efekte. Prvi, ki smo ga zaradi načina delovanja poimenovali PixelRain, omogoča kontrolirano ustvarjanje vizualnih artefaktov. Piksle, ki jih lahko izberemo na podlagi barve, kopira navzdol in s tem prepisuje druge piksle. S pomočjo tehnik sestavljanja vizualnih učinkov,

ki jih ponuja AE, lahko sledem tudi spremenimo smer.

Drugi vtičnik je poimenovan ElStAberr. Inspiriral ga pojav, imenovan elektrostatični odklon (ang. *electro static aberration*). Deluje tako, da premika piksle stran alin proti izbranemu centru učinka, s tem da močneje deluje na svetlejše piksle. Z njim lahko dosežemo različne vizualne rezultate, na primer ustvarimo iluzijo reliefnega tiska ali oblikujemo slike s pomočjo pazljivega nadzora nad svetlostjo delov slike.

Tretji vtičnik smo poimenovali MirrorMosaic, saj sestavi platno iz večih kopij videa, ki so zrcaljnene, tako da dobimo vtis simetrije. Vtičnik različne kopije videa zamakne v času. Z njegovo pomočjo lahko ustvarjamo različne vdeo tapete in animirane vzorce. Namenjen je delu videa, ki se ga lahko neopazno zanka.

Razvoj digitalnih vizualnih učinkov je zanimiv tako s tehnološkega vidika kot z izraznega. Pomembno je, da digitalni vizualni učinki niso končni cilj ampak le orodje, ki ga uporabljamo za oblikovanje digitalnega videa. Veselim se dela z vtičniki, ki sem jih razvil, tudi s tega vidika - ne le razvoja, temveč tudi prihodnje uporabe, saj so vsi razviti vtičniki uporabni in jih bom vprihodnje naprej razvijal in tudi uporabljal.

Primerjava z vtičniki vgrajenimi v AE je pokazala, da se izvajajo približno enako hitro in zasedejo primerljivo količino pomnilnika. Eksplicitno se z optimizacijo nisem ukvarjal, sem pa pri implementaciji ves čas imel hitrost izvajanja v mislih. Pri razvoju programske opreme, ki se ukvarja z videom, je pravilno delo s pomnilnikom ključnega pomena, saj je video po naravi pomnilniško zahteven. Vsaka napaka pri delu s pomnilnikom se je med razvojem pokazala med uporabo kot izjemno počasno delovanje ali pa sesutje vtičnika, programa in v redkih primerih celotnega sistema že po nekaj obdelanih sličicah.

Delo z AE SDK je zanimivo, med drugim zato, ker ima zelo veliko podporo za podporo starejših verzij programa AE. Tako lahko skozi uporabo AE SDK vidimo tudi razvoj programa gostitelja skozi leta. Vse, ki jih razvoj vtičnikov za AE zanima, pa bi rad opozoril na to, da je priročnik [3] mestoma

zastarel in celo napačen. Marsikateri podatek manjka in veliko sem odkril eksperimentalno.

Celotna programska koda je na voljo na naslovu `https://github.com/majthehero/glitchFXdev` in prek programa Git na naslovu `https://github.com/majthehero/glitchFXdev.git`

Literatura

[1] Pascal Binitzer. *Slepo Polje*. ŠKUC, Znanstveni inštitut Filozofke fakultete, Ljubljana, Slovenija, 1985.

[2] Barbara Borčič. *VIDEODOKUMENT: videoumetnost v slovenskem prostoru 1969-1998*. Zavod za odprto družbo - Slovenija, Ljubljana, Slovenija, 1999.

[3] Russell Belfer et al. After effects cc 2017.1 sdk guide, release 1, 2017.

[4] Marina Gržinić. *Rekonstruirana fikcija: novi mediji, (video) umetnost, postsocializem in retroavantgarda: teorija, politika, estetika: 1997-1985*. Koda. Študentska organizacija univerze, Študentska založba, Ljubljana, Slovenija, 1997.

[5] B. Hill, Th. Roger, and F. W. Vorhagen. Comparative analysis of the quantization of color spaces on the basis of the cielab color-difference formula. *ACM Trans. Graph.*, 16(2):109–154, April 1997.

[6] M. Jago. *Adobe Premiere Pro CC Classroom in a Book (2017 release)*. Classroom in a Book. Pearson Education, 2017.

[7] L. Manovich. *The Language of New Media*. Leonardo (Series) (Cambridge, Mass.). MIT Press, 2001.

[8] Nenad Puhovski. Jedan pomalo tužan pogled na američki video 1983. godine. In Žarana Papić, editor, *Videosfera*, pages 42–45. Studentski izdavački centar UKSSO Beograda, Beograd, Srbija, 1986.

[9] Thiadmer Riemersma. Colour metric, Apr 2012.

[10] Visual Effects Society. *The VES Handbook of Visual Effects*. Esevier Inc., Oxford, Anglija, 2010.

[11] Adobe Creative Team. *Adobe After Effects CC Classroom in a Book*. Adobe Press, 1st edition, 2013.

[12] Tom Duff Thomas Porter. Compositing digital images. *Computer Graphics SIGGRAPH'84*, 18(3):253–259, 1984.

[13] Nikolai Waldman. Math behind colorspace conversions, rgb-hsl, Aug 2017.

[14] Tatjana Zrimec. *Računalništvo in uporabniška programska oprema*. Fakulteta za računalništvo in informatiko, Ljubljana, Ljubljana, Slovenija, 1997.